SANDIA REPORT

SAND2016-XXXX
Unlimited Release
Printed March 2016

Modeling Bilevel Programs in Pyomo

William E. Hart, Richard Chen, John D. Siirola, Jean-Paul Watson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2016-XXXX
Unlimited Release
Printed March 2016

Modeling Bilevel Programs in Pyomo

William E. Hart, Richard L. Chen, John D. Siirola, Jean-Paul Watson
Sandia National Laboratories
PO Box 5800, MS 1318
Albuquerque, NM 87185
{wehart,rlchen,jdsiiro,jwatson}@sandia.gov

Abstract

We describe new capabilities for modeling bilevel programs within the Pyomo modeling software. These capabilities include new modeling components that represent subproblems, modeling transformations for re-expressing models with bilevel structure in other forms, and optimize bilevel programs with meta-solvers that apply transformations and then perform optimization on the resulting model. We illustrate the breadth of Pyomo's modeling capabilities for bilevel programs, and we describe how Pyomo's meta-solvers can perform local and global optimization of bilevel programs.

Acknowledgment

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Contents

1 Introduction

Many planning situtations involve the analysis of several objectives that reflect a hierarchy of decision-makers. For example, policy decisions are made at different levels of a government, each of which has a different objective and decision space. Similarly, robust planning against adversaries is often modeled with a 2-level hierarchy, where the defensive planner makes decisions that account for adversarial response.

Multilevel optimization techniques partition control over decision variables amongst the levels. Decisions at each level of the hierarchy may be constrained by decisions at other levels, and the objectives for each level may account for decisions made at other levels. In practice, multilevel problems have proven difficult to solve, and most of the literature has focused on *bilevel programs*, which model a 2-level hierarchy [2].

Although multilevel problems arise in many applications, few algebraic modeling languages (AML) have integrated capabilities for expressing these problems. AMLs are high-level programming languages for describing and solving mathematical problems, particularly optimization-related problems [9]. AMLs provide a mechanism for defining variables and generating constraints with a concise mathematical representation, which is essential for large-scale, real-world problems that involve thousands or millions of constraints and variables. GAMS [5], YALMIP [10] and Pyomo provide explicit support for modeling bilevel programs. A variety of other AMLs support the solution of bilevel programs through the expression of KKT conditions and associated reformulations using mixed-complementarity conditions, but these reformulations must be expressed by the user in these AMLs.

In this paper, we describe new functionality in Pyomo 4.3 for expressing and optimizing multilevel models in the Pyomo modeling environment. Pyomo is an open-source software package that supports the definition and solution of optimization applications using the Python language [12, 11, 7, 8]. Python is a powerful programming language that has a clear, readable syntax and intuitive object orientation. Pyomo uses an object-oriented approach for defining models that contain decision variables, objectives, and constraints.

The main point of this paper is to demonstrate that Pyomo provides an intuitive syntax for expressing multilevel optimization problems. Multilevel models can be easily expressed with Pyomo modeling components for submodels, which can be nested in a general manner. Further, Pyomo's object-oriented design naturally supports the ability to automate the reformulation of multilevel models into other forms. In particular, we describe Pyomo's capabilities for transforming bilevel models for several broad classes of problems. We describe Pyomo meta-solvers that transform bilevel programs into mixed integer progrms (MIP) or nonlinear programs (NLP), which are then optimized with standard solvers.

The remainder of this paper is organized as follows. Section 2 describes motivating classes of bilevel programming problems. Section 3 describes how Pyomo supports modeling of submodels. Sections 4 and 5 describe transformations that automate the reformulation of bilevel programs and meta-solvers in Pyomo that leverage these transformations to support global and local optimiza-

tion of bilevel programs. Section 6 discusses Pyomo's ability to model multilevel problems, and possible extensions for non-cooperative bilevel programs.

2 Motivating Problems

In multilevel optimization problems, a subset of decision variables at each level is constrained to take values associated with an optimal solution of a distinct, lower level optimization problem. For example, a general formulation for bilevel programs is

$$\begin{array}{ll} \min_{x \in X, y} & F(x,y) \\ \text{s.t.} & G(x,y) \leq 0 \\ & y \in P(x) \end{array} \tag{1}$$

where

$$P(x) \quad = \quad \arg\min_{y \in Y} \quad \begin{array}{l} f(x,y) \\ g(x,y) \leq 0 \end{array}$$

$P(x)$ defines a *lower-level problem*, which may have multiple solutions. Here x is the primary upper-level decision, and y is the anticipated lower-level decision.

When $P(x)$ contains multiple solutions, this formulation ensures that the selected value for y will minimize $F(x,y)$. Consequently, this formulation has been called *optimistic* or *cooperative*, since the selection of the lower-level decision variables minimized the upper-level objective. Most research on bilevel programming has considered this problem, since this formulation has an optimal solution under reasonable assumptions.

The following subsections describe specializations of Equation (1) that will be explored in greater detail throughout this paper. These are well-studied classes of bilevel programs that that can be reformulated into other canonical optimization forms.

2.1 Linear Bilevel Programs with Continuous Variables

Multilevel linear programming considers the case where decision variables are continuous, and both objectives and contraints are linear. In the 2-level case, we have the following linear bilevel program:

$$\begin{array}{lll} \min_{x,y} & c_1^T x + d_1^T y & \\ \text{s.t.} & A_1 x + B_1 y \leq b_1 & \\ & x \geq 0 & \\ & \min_y & c_2^T x + d_2^T y \\ & & A_2 x + B_2 y \leq b_2 \\ & & y \geq 0 \end{array} \tag{2}$$

2.2 Quadratic Min/Max

Consider the case where the lower-level decisions y do not constrain the upper-level decisions. Let

$$X = \{x \mid A_1 x \leq b, x \geq 0\}$$

Then we have

$$\begin{array}{lll} \min_{x \in X} & \max_{y \geq 0} & c_1^T x + d_1^T y + x^T Q y \\ & & A_2 x + B_2 y \leq b_2 \end{array} \tag{3}$$

In our discussion below, we allow the x_i to be binary without loss of generality.

3 Modeling Bilevel Programs

3.1 Pyomo Models

Pyomo supports an object-oriented design for the definition of optimization models. The basic steps of a simple modeling process are as follows:

1. Create model object and declare components
2. Instantiate the model
3. Apply solver
4. Interrogate solver results

In practice, these steps may be applied repeatedly with different data or with different constraints applied to the model. However, we focus on this simple modeling process throughout this paper.

A Pyomo *model* consists of a collection of modeling *components* that define different aspects of the model. Pyomo includes the modeling components that are commonly supported by modern AMLs: components for decision variables, objectives and constraints are used to express the model, and components for index sets and symbolic parameters are used to manage data used in model expressions. These modeling components are defined in Pyomo through the following Python classes:

`Var`	decision variables in a model
`Objective`	expressions that are minimized or maximized in a model
`Constraint`	constraint expressions in a model
`Set`	set data that is used to define a model instance
`Param`	parameter data that is used to define a model instance

Modeling components are directly added to a Pyomo model object as an attribute of the object. For example:

```
model = ConcreteModel()
model.x = Var()
```

Pyomo defines two model classes: `ConcreteModel` and `AbstractModel`. In concrete models, components are immediately initialized when they are added to the model. In abstract models, components are explicitly initialized later, which allows the creation of multiple model instances from a single abstract model.

Another Pyomo component is the `Block` class, which defines a named collection of model components. Blocks provide a convenient mechanism for segregating different elements of a model within separate namespaces, and they can be used to define repetitive model structure in a concise manner. For example, the following model defines two variables `x`:

```
model = ConcreteModel()
model.x = Var()
model.b = Block()
model.b.x = Var()
```

Block `b` defines a namespace that allows these two variables to be uniquely referenced.

3.2 Modeling Bilevel Programs

The `pyomo.bilevel` package extends Pyomo by defining a new modeling component: `SubModel`. The `SubModel` component defines a subproblem that represents the lower level decisions in a bilevel program. This component is like a `Block` component; any components can be added to a `SubModel` object. In general, a submodel is expected to have an objective, one or more variables and it may define constraints.

The `SubModel` class generalizes the `Block` component by including constructor arguments that denote which variables in the submodel should be considered fixed or variable. When expressions in a submodel refer to variables defined outside of the submodel, the user needs to indicate whether these are fixed values defined by an upper-level problem. Fixed variables are treated as constants within the submodel, but non-fixed variables are defined by the current submodel or by a lower-level problem.

Consider the following example:

$$\begin{array}{lll} \min_{x,y,v} & x+y+v & \\ \text{s.t.} & x+v \geq 1.5 & \\ & 1 \leq x \leq 2 & \\ & 1 \leq v \leq 2 & \\ & \max_{y,w} & x+w \\ & & y+w \leq 2.5 \\ & & 1 \leq y \leq 2 \\ & & 1 \leq w \leq 2 \end{array} \tag{4}$$

The following Pyomo model defines four variables, `x`, `v`, `sub.y` and `sub.w`:

```
from pyomo.environ import *
from pyomo.bilevel import *

model = ConcreteModel()
model.x = Var(bounds=(1,2))
model.v = Var(bounds=(1,2))
model.sub = SubModel()
model.sub.y = Var(bounds=(1,2))
model.sub.w = Var(bounds=(-1,1))

model.o = Objective(expr=model.x + model.sub.y + model.v)
model.c = Constraint(expr=model.x + model.v >= 1.5)
model.sub.o = Objective(expr=model.x+model.sub.w, sense=maximize)
model.sub.c = Constraint(expr=model.sub.y + model.sub.w <= 2.5)
```

Variables `x` and `v` are declared in the upper-level problem, and `v` only appears in the upper-level problem. Variables `sub.y` and `sub.w` are declared in the submodel. However, note that the `sub.y` variable appears in the upper-level problem, while the `sub.w` variable only appears in the lower-level problem.

These modeling capabilities are quite general. Expressions in submodels can be linear or nonlinear, convex or nonconvex, continuous or discontinuous, and more. Additionally, submodels can be nested to an arbitrary degree. Thus, the range of bilevel programs that can be expressed with Pyomo is quite broad. However, the real challenge is solving these models. The following sections describe two general strategies for solving the two classes of bilevel programs described in Section 2.

In each section we describe model transformations and their use in meta-solvers. Pyomo's object-oriented design supports the structured transformation of models. Pyomo can iterate through model components as well as nested model blocks. Thus, model components can be easily transformed locally, and global data can be collected to support global transformations. Further, Pyomo components and blocks can be activated and deactivated, which facilitates *in place* transformations that do not require the creation of a separate copy of the original model.

4 Solving Linear Bilevel Programs

We consider the formulation for linear bilevel programs described in Equation (2):

$$\begin{array}{lll} \min_{x,y} & c_1^T x + d_1^T y & \\ \text{s.t.} & A_1 x + B_1 y \leq b_1 & \\ & x \geq 0 & \\ & \min_y & c_2^T x + d_2^T y \\ & & A_2 x + B_2 y \leq b_2 \\ & & y \geq 0 \end{array} \tag{2}$$

Following Bard [1], we can replace the lower-level problem with corresponding optimality conditions. This transformation gives the following model:

$$\begin{array}{ll} \min & c_1^T x + d_1^T y \\ \text{s.t.} & A_1 x + B_1 y \leq b_1 \\ & d_2 + B_2^T u - v = 0 \\ & b_2 - A_2 x - B_2 y \geq 0 \perp u \geq 0 \\ & y \geq 0 \perp v \geq 0 \\ & x \geq 0, y \geq 0 \end{array} \tag{5}$$

This transformation results in a mathematical program with equillibrium constraints (MPEC).

Pyomo's `pyomo.bilevel` package automates the application of this model transformation. For example, if `model` defines an linear bilevel program, then the code applies this transformation to change the model in place:

```
xfrm = TransformationFactory('bilevel.linear_mpec')
xfrm.apply_to(model)
```

The `bilevel.linear_mpec` transformation modifies the model by creating a new block of variables and constraints for the optimality conditions in the lower-level problem.

A variety of solution strategies can leverage this transformation to support optimization. Pyomo defines two meta-solvers, which apply the `bilevel.linear_mpec` transformation and then perform optimization with a third-party solver.

Consider the following problem, which is Example 5.1.1. in Bard [1]:

$$\begin{array}{lll} \min_{x,y} & x - 4y & \\ & x \geq 0 & \\ \text{s.t.} & \min_y y & \\ & \text{s.t.} & -x - y \leq -3 \\ & & -2x + y \leq 0 \\ & & 2x + y \leq 12 \\ & & 3x - 2y \leq 4 \end{array} \tag{6}$$

Note that the last constraint in this example is negated from the text in Bard [1]; this corrects an error in the example, which is reflected in Bard's discussion of the solution to this example. The Pyomo model for this problem is:

```
# bard511.py

from pyomo.environ import *
from pyomo.bilevel import *

M = ConcreteModel()
M.x = Var(bounds=(0,None))
M.sub = SubModel()
M.sub.y = Var(bounds=(0,None))

M.o = Objective(expr=M.x - 4*M.sub.y, sense=minimize)

M.sub.o = Objective(expr=M.sub.y, sense=minimize)
M.sub.c1 = Constraint(expr=-  M.x -   M.sub.y <= -3)
M.sub.c2 = Constraint(expr=-2*M.x +   M.sub.y <=  0)
M.sub.c3 = Constraint(expr= 2*M.x +   M.sub.y <= 12)
M.sub.c4 = Constraint(expr= 3*M.x - 2*M.sub.y <=  4)

model = M
```

4.1 Global Optimization

Following Fortuny-amat and McCarl [4], Pyomo's `bilevel_blp_global` meta-solver chains together reformulations to generate the following sequence of models:

$$\text{BLP} \Rightarrow \text{MPEC} \Rightarrow \text{GDP} \Rightarrow \text{MIP},$$

where GDP refers to generalized disjunctive programs. Note that this leverages advanced modeling capabilities in Pyomo for MPEC [6] and GDP [13].

Pyomo provides general support for solving MIPs with commercial and open-source solvers. The `bilevel_blp_global` meta-solver applies these transformations, solves the resulting MIP, and translates the MIP solution back into the original linear bilevel program. The result is a globally optimal solution. For example, the `pyomo` command can be used to execute the `mpec_blp_global` solver using a specified MIP solver:

```
pyomo solve --solver=bilevel_blp_global \
            --solver-options="solver=glpk" bard511.py
```

Note that a "Big-M" transformation is used to convert the GDP model into a MIP. The default M value is very large, which may make it difficult to solve the resulting MIP problem. Hence, this solver includes a `bigM` option that can be used to specify a problem-specific value:

```
pyomo solve --solver=bilevel_blp_global \
            --solver-options="bigM=100 solver=glpk" bard511.py
```

4.2 Local Optimization

Pyomo's `bilevel_blp_local` meta-solver chains together reformulations to generate the following sequence of models:

$$\text{BLP} \Rightarrow \text{MPEC} \Rightarrow \text{NLP}.$$

This leverages model transformations in `pyomo.gdp` to transform an MPEC into an NLP through a simple nonlinear transformation adapted from Ferris et al. [3]. For example, the complementarity condition

$$y \geq 0 \perp w \geq 0$$

is transformed to the constraints

$$\begin{aligned} y &\geq 0 \\ w &\geq 0 \\ yw &\leq \varepsilon. \end{aligned}$$

Pyomo provides general support for solving NLPs with commercial and open-source solvers. The `bilevel_blp_local` meta-solver applies these transformations, solves the resulting NLP, and translates the solution back into the original linear bilevel program. In general, the result is a locally optimal solution. For example, the `pyomo` command can be used to execute the `mpec_blp_local` solver using a specified NLP solver:

```
pyomo solve --solver=bilevel_blp_local \
            --solver-options="solver=ipopt" bard511.py
```

Note that the tolerance value ε is initially set to a small value, which some solvers may have difficulty with. This value can be explicitly set with the `mpec_bound` option:

```
pyomo solve --solver=bilevel_blp_local \
            --solver-options="mpec_bound=0.01 solver=ipopt" bard511.py
```

5 Solving Quadratic Min-Max Bilevel Programs

We consider the formulation for quadratic min-max bilevel programs described in Equation (3):

$$\begin{array}{lll} \min_{x\in X} & \max_{y\geq 0} & c_1^T x + d_1^T y + x^T Q y \\ & & A_2 x + B_2 y \leq b_2 \end{array} \tag{3}$$

where

$$X = \{x \mid A_1 x \leq b, x \geq 0\}.$$

The lower-level problem is linear, so we can replace the lower-level problem with corresponding optimality conditions. Since, the objectives are opposite and the upper-level constraints do not constrain the lower level decisions, we get a single minimization problem. This transformation gives the following model:

$$\begin{array}{ll} \min & c_1^T x + (b_2 - A_2 x)^T \nu \\ \text{s.t.} & B_2^T \nu \geq d_1 + Q^T x \\ & A_1 x \leq b_1 \\ & x \geq 0, \nu \geq 0 \end{array} \tag{7}$$

In Pyomo, this is implemented as the `bilevel.linear_dual` transformation.

If $A_2 \equiv 0$, then the lower-level problem does not constrain the upper-level decisions. This is a simple case, where the transformation generates a linear program if the upper-level decision variables x are continuous and a MIP if some or all of the x are binary.

More generally, suppose that the upper-level decision variables x are binary and $A_2 \not\equiv 0$. We can linearize the quadratic terms in the objective using `gdp.bilinear` transformation, which creates the following disjunctive representation:

$$\begin{array}{ll} \min & c_1^T x + b_2^T \nu - 1^T z \\ \text{s.t.} & B_2^T \nu \geq d_1 + Q^T x \\ & A_1 x \leq b_1 \\ & \begin{pmatrix} x_i = 0 \\ z_i = 0 \end{pmatrix} \wedge \begin{pmatrix} x_i = 1 \\ z_i = A_2^T(*, i)\nu \end{pmatrix} \\ & x_i \geq \{0,1\}, \nu \geq 0 \end{array} \tag{8}$$

Subsequently, this GDP can be transformed into a MIP using a Big-M transformation.

Consider the following network interdiction problem, for which an attacker eliminates links in a network to minimize the maximum flow through the network from a fixed source s to a fixed destination t. Let N be the nodes in the network through which flow occurs. Let y_{ij} be a variable that indicates flow from node i to node j, and let c_{ij} be the maximum capacity on that arc. The attacker selects b variables x_{ij}; if x_{ij} is one then the arc is removed. This network interdiction problem can be written as:

$$\begin{array}{lll} \min_{x\in X} \quad \max_{y} & \eta & \\ & \sum_{i\in N} y_{in} = \sum_{j\in N} y_{nj} \quad \forall n \in N & \text{(flow balance constraint)} \\ & \eta \leq \sum_{j\in N} y_{sj} & \text{(node s flow)} \\ & \eta \leq \sum_{i\in N} y_{it} & \text{(node t flow)} \\ & 0 \leq y_{ij} \leq t_{ij}(1 - x_{ij}) \quad \forall \text{ arcs } (i,j) & \text{(capacity constraint)} \end{array} \tag{9}$$

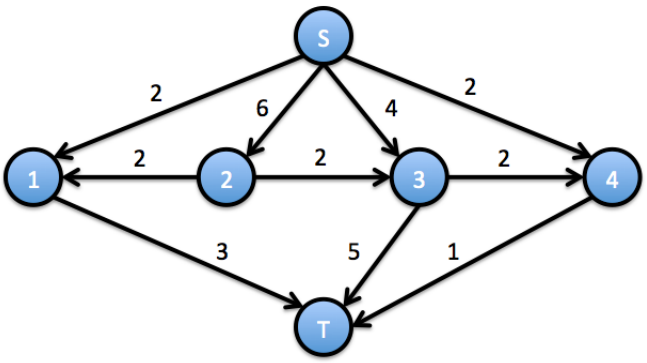
where

$$X = \left\{ x \mid x_{ij} \in \{0,1\}, \sum_{i,j} x_{ij} \leq b \right\}.$$

The following Pyomo model describes this bilevel program:

```
# interdiction.py

from pyomo.environ import *
from pyomo.bilevel import *
from interdiction_data import A, budget

M = ConcreteModel()
M.x = Var(A.keys(), within=Binary)
M.budget = Constraint(expr=summation(M.x) <= budget)
M.sub = SubModel()
M.sub.f = Var()
M.sub.y = Var(A.keys(), within=NonNegativeReals)

# Min/Max objectives
M.o = Objective(expr=M.sub.f, sense=minimize)
M.sub.o  = Objective(expr=M.sub.f, sense=maximize)

# Flow constraint
def flow_rule(M, n):
    return sum(M.y[i,n] for i in sequence(0,4) if (i,n) in A) == \
           sum(M.y[n,j] for j in sequence(1,5) if (n,j) in A)
M.sub.flow = Constraint(sequence(1,4), rule=flow_rule)

# Source constraint
def s_rule(M):
    model = M.model()
    return model.sub.f <= sum(M.y[0,j] for j in sequence(1,4) if (0,j) in A)
M.sub.s = Constraint(rule=s_rule)

# Destination constraint
def t_rule(M):
    model = M.model()
    return model.sub.f <= sum(M.y[j,5] for j in sequence(1,4) if (j,5) in A)
M.sub.t = Constraint(rule=t_rule)

# Capacity constraint
def c_rule(M, i, j):
    model = M.model()
    return M.y[i,j] <= A[i,j]*(1-model.x[i,j])
M.sub.c = Constraint(A.keys(), rule=c_rule)

model = M
```

In this example, the file `interdiction_data.py` defines a simple network with 6 nodes (including *s* and *t*), adapted from an example by Will Traves:



Pyomo's `bilevel_ld` meta-solver applies the `bilevel.linear_dual` transformation and then applies subsequent transformations when $A_2 \neq 0$.

```
pyomo solve --solver=bilevel_ld\
            --solver-options="bigM=100 solver=glpk" interdiction.py
```

Note that this solver includes a `bigM` option that can be used to specify a problem-specific value when a MIP is generated from a GDP.

6 Discussion

Note that Pyomo's ability to model multilevel optimization problem extends far beyond the bilevel programs that are currently supported. For example, declarations of `SubModel` can be arbitrarily nested with clear semantics. For example, consider the following trilevel model [14]:

$$\begin{array}{ll} \min_{x,y,z} & x-4y+2z \\ \text{s.t.} & -x-y \leq -3 \\ & -3x+2y-z \geq -10 \end{array}$$

$$\begin{array}{ll} \min_{y,z} & x+y-z \\ \text{s.t.} & -2x+y-2z \leq -1 \\ & 2x+y+4z \leq 14 \end{array}$$

$$\begin{array}{ll} \min_{z} & x-2y-2z \\ \text{s.t.} & 2x-y-z \leq 2 \end{array}$$

The following model illustrates how this trilevel model could be implemented in Pyomo:

```
from pyomo.environ import *
from pyomo.bilevel import *

M = ConcreteModel()
M.x = Var()
M.s = SubModel()
M.s.y = Var()
M.s.s = SubModel()
M.s.s.z = Var()

M.o = Objective(expr=       M.x - 4*M.s.y + 2*M.s.s.z)
M.c1 = Constraint(expr=   - M.x -   M.s.y             <=  -3)
M.c2 = Constraint(expr=  -3*M.x + 2*M.s.y             >= -10)
M.s.o = Objective(expr=     M.x +   M.s.y -   M.s.s.z)
M.s.c1 = Constraint(expr=-2*M.x +   M.s.y - 2*M.s.s.z <=  -1)
M.s.c2 = Constraint(expr= 2*M.x +   M.s.y + 4*M.s.s.z <=  14)
M.s.s.o = Objective(expr=   M.x - 2*M.s.y - 2*M.s.s.z)
M.s.s.c = Constraint(expr=2*M.x -   M.s.y -   M.s.s.z <=   2)

model = M
```

Pyomo use of object-oriented model specification makes it fundamentally different from the specification of bilevel models in GAMS and YALMIP. Both GAMS and YALMIP allow users to specify expressions for variables, objectives and constraints, and then the users specifies which of these are associated with an upper-level or lower-level problem. This design allows users to mix-and-match different modeling components in a flexible manner. However, it is limited to a strictly bilevel form. By contrast, Pyomo submodels can be nested in an arbitrary manner. This includes multilevel models, as was just illustrated. But it also allows for the specification of a tree of nested submodels. For example, Pyomo supports the specification of independent submodels at the same level, which can be used to model a single agent cooperating with decisions for two independent agents that make subsequent decisions..

Earlier, we noted that Pyomo supports optimistic or cooperative bilevel models. It seems relatively straightforward to extend Pyomo's current semantics to support pessimistic bilevel models. For example, a `pessimistic` constructor option could be added to the `SubModel` component. This capability will be added when transformations and/or solvers are added to Pyomo for these bilevel models.

References

[1] John F. Bard. *Practical bilevel optimization: Algorithms and applications*. Kluwer Academic Publishers, Dordrecht, 1998.

[2] Colson Benoîit, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Ann Oper Res*, 153:235–256, 2007.

[3] Michael C. Ferris, Steven P. Dirkse, and A. Meeraus. Mathematical programs with equilibrium constraints: Automatic reformulation and solution via constrained optimization. In T. J. Kehoe, T. N. Srinivasan, and J. Whalley, editors, *Frontiers in Applied General Equilibrium Modeling*, pages 67–93. Cambridge University Press, 2005.

[4] José Fortuny-Amat and Bruce McCarl. A representation and economic interpretation of a two-level programming problem. *The Journal of the Operations Research Society*, 32(9): 783–792, 1981.

[5] GAMS. GAMS home page. http://www.gams.com, 2008.

[6] William E. Hart and John D. Siirola. Modeling mathematical programs with equilibrium constraints in pyomo. Technical Report SAND2015-5584, Sandia National Laboratories, July 2015.

[7] William E. Hart, Jean-Paul Watson, and David L. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3), 2011.

[8] William E. Hart, Carl Laird, J.-P. Watson, and David L. Woodruff. *Pyomo: Optimization Modeling in Python*. Springer, 2012.

[9] Josef Kallrath. *Modeling Languages in Mathematical Optimization*. Kluwer Academic Publishers, 2004.

[10] Johan Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *2004 IEEE Intl Symp on Computer Aided Control Systems Design*, 2004.

[11] Pyomo Home. Pyomo home page. https://www.pyomo.org, 2015.

[12] Pyomo Software. Pyomo software trac site. https://software.sandia.gov/svn/public/pyomo/, 2015.

[13] John D. Siirola. Modeling generalized disjunctive programs in pyomo. Technical report, Sandia National Laboratories, 2015. (in preparation).

[14] Guangquan Zhang, Jie Lu, Javier Montero, and Yi Zeng. Model, solution concept and Kth-best algorithm for linear trilevel programming. *Information Sciences*, 180:481–492, 2010.

DISTRIBUTION:

1 MS 0899 Technical Library, 9536 (electronic copy)


Sandia National Laboratories